Breaking Encryption: Externer-Noten-Client

Marlon Starkloff

Februar 2023

Der Externer-Noten-Client oder auch ENC genannt, ist ein Programm, welches vom Hessischen Kultusministerium für die Lehrer- und Schülerdatenbank (LUSD) entwickelt wurde. Es dient zur einfachen Bearbeitung von Schülerdaten, wie beispielsweise Zeugnisnoten oder Bemerkungen. Nach längerer Analyse verschiedener Versionen des ENCs wurden Sicherheitslücken in Bezug auf die Verschlüsselung der exportierten Dateien gefunden.

Dieses Dokument bezieht sich auf die aktuellste Version und soll auf die fehlerhafte Verschlüsselung hinweisen sowie Orientierungspunkte geben, wie man sie robuster gestalten kann.

1 Externer-Noten-Client

Der ENC kann von der Weboberfläche des hessischen Schulportals heruntergeladen und lokal gestartet werden. Zugleich kann man die benötigte und verschlüsselte Schülerdatei herunterladen. Nachdem der ENC gestartet wurde, wird man aufgefordert, ein Passwort einzugeben. Dieses wird benötigt, um die heruntergeladene Datei zu entschlüsseln. Das Passwort wird einem auf der Weboberfläche nach dem Download angezeigt. Sobald die Bearbeitung abgeschlossen ist, wird die Datei wieder über die Weboberfläche hochgeladen. Danach kann sie in die LUSD eingespielt werden.

Wie schon angedeutet, beinhalten diese Schülerdateien sensible und schulbezogene Daten. Daher ist es umso wichtiger sie sicher zu verschlüsseln, damit Unbefugte, welche Zugang zu diesen Dateien erlangt haben, sie nicht auslesen können. Die Möglichkeit diese Dateien zu entschlüsseln, ohne das Passwort zu kennen, stellt ein Risiko für den Datenschutz der Schüler dar.

1.1 Entschlüsselung

Um die Sicherheitslücke besser nachvollziehen zu können, ist es wichtig zu verstehen, wie die Entschlüsselung im Detail abläuft. Das Passwort, welches man auf der Weboberfläche angezeigt bekommt und man im ENC eingeben muss, folgt immer einer bestimmten Struktur. Es ist 8 Zeichen lang und besteht aus Buchstaben, Zahlen und Sonderzeichen. Nachdem das Passwort in den ENC eingetragen wurde, wird es in eine bestimmte Funktion übergeben. Diese sieht wie folgt aus:

```
public static string[] GenUnPackKey(string pw) {
   StringBuilder sb = new StringBuilder();
   StringBuilder sb2 = new StringBuilder();
   for (int i = 0; i < pw.Length; i++) {
      byte num = ZeichentauschEntschluesseln(pw[i]) - 48;
      char value = (char)((num & 7) + 48);
      sb.Append(value);
      value = (char)(((num & 0xF8) >> 3) + 65);
      sb2.Append(value);
   }
   return new string[2] { sb.ToString(), sb2.ToString() };
}
```

Die Funktion akzeptiert als Parameter das Passwort und gibt einen String Array zurück. In der Funktion durchläuft das Passwort verschiedene bitweiser Operationen und wird in zwei Strings aufgeteilt. Die genaue Funktionsweise ist irrelevant, stattdessen ist die Struktur der Rückgabewerte interessant. Um den Schlüssel für die Entschlüsselung herzuleiten, wird PBKDF2 mit 400.000 Iterationen und SHA-256 benutzt. Als Startwert wird der erste Wert aus dem zurückgegebenen Array genutzt.

Der Vorgang kann vereinfacht wie folgt dargestellt werden:

```
string password = "ABCD1234";
string[] password_array = GenUnPackKey(password);
byte[] key = pbkdf2(password_array[0], 400000, SHA256);
```

Anschließend wird der hergeleitete Schlüssel zum Entschlüsseln der Datei genutzt. Die Verschlüsselung welche verwendet wird, ist AES-256 CBC.

1.2 Schwachstelle der Verschlüsselung

Die Schwachstelle, welche die Verschlüsselung angreifbar macht, ist die Funktion `GenUnPackKey` und dessen Rückgabewert. Der erste Wert im Array, welcher für PBKDF2 benutzt wird, ist unsicher. Wie man es an diesem Beispiel sehen kann:

```
string[] passwords = GenUnPackKey("dk_DSm5G");
Console.Write(passwords[0]);
```

Dieser Code würde `43743557` zurückgeben. Es ist kein Zufall, dass der Wert nur aus Zahlen zwischen 0 und 7 besteht sowie 8 Zeichen lang ist. Die Funktion `GenUnPackKey` wird immer einen Wert mit dieser Struktur zurückgeben.

Dies wird auch nochmal bestätigt, wenn man sich den Code anschaut, welcher kurz vor dem Aufruf von PBKDF2 ausgeführt wird:

```
public static bool IsPwValid(string pw) {
   if (new Regex("[0-7]{" + pw.Length + "}").IsMatch(pw)) {
      return pw.Length == 8;
   }
   return false;
}
```

Dieser Code überprüft die schon oben genannten Bedingungen. Das Passwort, welches in PBKDF2 übergeben wird, muss:

- Aus Ziffern zwischen 0 und 7 bestehen
- 8 Zeichen lang sein

Anhand dieser Informationen kann man herleiten, dass die Anzahl an möglichen Passwörtern 8^8 oder 16.777.216 ist. Unabhängig davon, wie kompliziert das ursprüngliche Passwort war.

2 Exploitation

In der Theorie wäre es nun möglich alle 8^8 Kombinationen zu testen bis der richtige Schlüssel gefunden wurde. Diese Methode dauert allerdings sehr lange und ist nicht die effizienteste, besonders dadurch, dass PBKDF2 400.000 Iterationen pro Schlüssel durchführt. Stattdessen sollte man einen Rainbow Table generieren.

2.1 Rainbow Table

Möchte man mehrere Schülerdateien entschlüsseln, müsste man die zeit- und ressourcenintensiven Rechnungen für jede Datei erneut durchführen. Um dies zu vermeiden, gibt es sogenannte Rainbow Tables. Bei denen alle möglichen Kombinationen, welche berechnet werden können, in eine Datei geschrieben werden. Anschließend muss nur noch die Datei gelesen und jeder Schlüssel getestet werden. Diese Methode hat den Vorteil, dass die 8^8 Passwörter und die 400.000 Iterationen von PBKDF2 nur einmal berechnet werden müssen.

2.2 Realisierbarkeit

Obwohl die Anzahl der gesamten Iterationen enorm hoch ist, ist es dennoch möglich, mit moderner Hardware einen Rainbow Table in wenigen Stunden zu generieren. Mit einem optimierten Programm zur Generierung und auf einem durchschnittlich guten Laptop würde dies ungefähr 20 Stunden dauern.

2.3 Voraussetzungen

Um einen Angriff gegen eine Schülerdatei erfolgreich durchzuführen, gibt es nur eine Voraussetzung.

- Der Angreifer hat einmalig eine Kopie des ENC erhalten

Dies ist notwendig, da der ENC-Quellcode weder öffentlich noch für jemanden außer Lehrer zugänglich ist.

2.4 Proof of Concept

Ein einfaches Proof of Concept Skript existiert für diese Sicherheitslücke nicht. Um den Rainbow Table effizient genug generieren zu können, muss ein optimiertes Programm entwickelt werden.

Das bedeutet allerdings nicht, dass so ein Programm nicht existiert. Für die Vorbereitung dieses Reports wurde ein entsprechendes Programm für die Generierung und Überprüfung der Schlüssel entwickelt. Genutzt wurde dafür Rust, sowie eine schnellere Implementierung von PBKDF2 in C. Um den Prozess zu beschleunigen wurde außerdem Multithreading implementiert.

Wie schon beschrieben, liegt die Dauer der Generierung dadurch bei wenigen Stunden. Das Überprüfen der Schlüssel an sich dauert hingegen nur wenige Sekunden.

3 Auswirkungen

Die Möglichkeit willkürlich jede Schülerdatei zu entschlüsseln verletzt nicht nur den Datenschutz der Schüler, sondern ermöglicht dem Angreifer auch Änderungen an der Datei vorzunehmen und diese anschließend wieder zu verschlüsseln. Man nehme folgendes Szenario:

Ein Lehrer hat die Datei mit den Noten der Schüler im Klassenraum bearbeitet, aber noch nicht wieder hochgeladen. Nun wird er abgelenkt oder ist für kurze Zeit anderweitig beschäftigt. Sollte der Lehrer den Rechner nicht gesperrt haben, reichen nur wenige Sekunden. Durch einen Bad-USB könnte der Cracker heruntergeladen, die Notendatei gesucht, bearbeitet und wieder verschlüsselt werden.

Auch wenn es Handlungsempfehlungen mit dem Umgang des ENCs gibt, wie beispielsweise das Sperren des Rechners, wird es bei der Anzahl an Schulen und Lehrern immer jemanden geben, der diese Empfehlungen nicht komplett befolgt.

4 Handlungsempfehlungen

Es ist anzumerken, dass der folgende Vorschlag, wie man die Verschlüsselung besser gestalten kann, möglicherweise nicht auf das Bildungsnetz angepasst werden kann. Da keine genaue Auskunft besteht, wie die LUSD oder das Bildungsnetz funktioniert.

Es sollte definitiv davon abgesehen werden, die Funktion `GenUnPackKey` weiterhin zu verwenden. Generell sollte man keine eigenen Implementierungen in Bezug auf Kryptografie anstreben. Moderne kryptografische Funktionen wurden bereits gegen verschiedene Angriffe getestet und stellen eine höhere Sicherheit gegenüber eigenen Implementierungen dar.

Zurzeit wird als Funktion zur Herleitung der Schlüssel PBKDF2 verwendet. Eine bessere Alternative wäre Argon2, welches verschiedene Schwachpunkte von PBKDF2 beseitigt und Bruteforce Attacken erschwert.

Um Rainbow Table Angriffe zu vermeiden, darf der Salt für PBKDF2 bzw. Argon2 nicht statisch sein. Zurzeit ist der Salt im ENC immer gleich, für jede Schülerdatei. Das hat im Grunde die gleiche Wirkung, wie gar keinen Salt zu verwenden. Stattdessen sollte beim Export der Datei für diese einen zufälligen Salt generiert und diesen in den unverschlüsselten Header der Schülerdatei gespeichert werden.